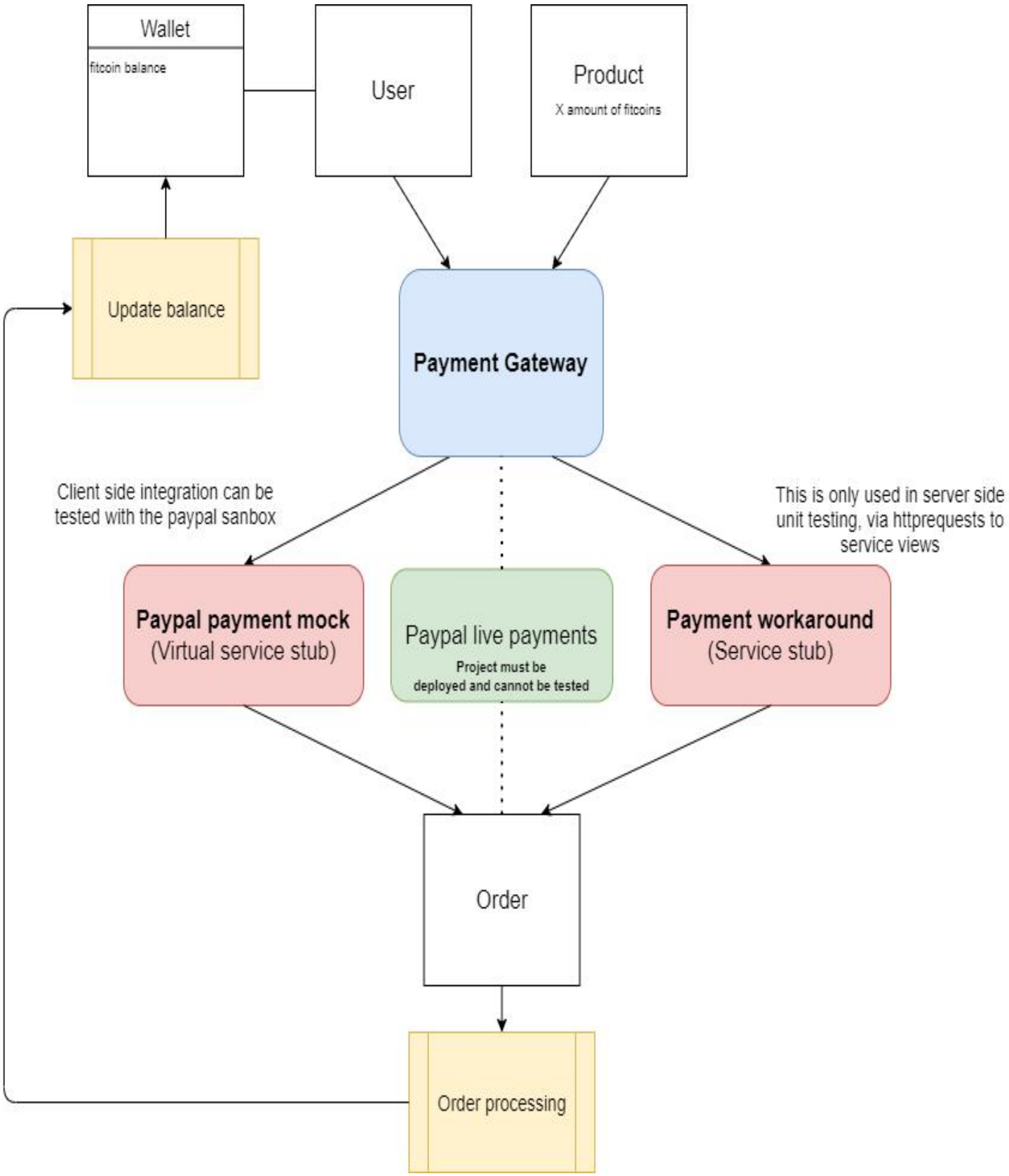
Design patterns

Fitcoin implementation



Service stub

See picture at top for reference

The intent of a service stub is to temporarily recreate an unavailable or unstable service with a (usually) hard-coded implementation that returns the same end result as the aforementioned service.

Our motivation for doing this, is to replace our paypal payment service during server unit testing with a different service (i.e Service stub) that runs locally, fast and in memory to improve our development experience.

To do this, while unit testing we bypass the paypal ‘‘payment_validation’’ controller view by calling a different controller view called ‘‘service_payment_validation’’. The service stub mimics the functionality of the paypal controller however it does not communicate with paypal at all. In addition to that, the service stub allows error insertions to mimic errors that could arise using paypal payments.

Mock service *(Additional design pattern)*

See picture at top for reference

Falls under the same category as the service stub in terms of intent.

Our motivation for implementing this design pattern, was the fact that we are able to perfectly mimic the paypal payment process through the interface, and thus, giving us a more complete development experience.

To test the payments through the interface, we have integrated a service provided by paypal, called paypal sandbox. With this integration we can successfully test the interface integration of paypal without deploying. This service is however not usable in server side unit testing, making this a mocking virtual service-stub.

Service types reference:
https://www.linkedin.com/pulse/what-difference-between-stub-mock-virtual-service-wojciech-bulaty

Gateway

See picture at top for reference

The intent of a gateway is to wrap all the special API code of external services into a class whose interface looks like a regular object. Other objects access the resource through this Gateway, which translates the simple method calls into the appropriate specialized API.

Our motivation to use this would be to simplify the usage of the external service, especially for repeated use and to make other team members lives easier.

The extremity to which the payment integration could be defined as a gateway, is subject for discussion. The sole reason for this is the fact that the paypal payment integration is mainly a client side integration making any attempts at a regular, class like object impossible. However the payment application as a whole could be viewed as a payment gateway. Making payments, in different manners can be accessed via only as much as an http request (url), which anyone can do, from any part of the project.

Money

See picture below for reference

The intent of money is quite simple, to represent a monetary value. That's to say in our case, represent something like a currency

Our motivation to do this was that we wanted to have our own currency on the website, in many ways similar to the steam market economy, where customers add balance to their steam account before using it to buy products on the website. For this we need a global exchange rate translation to our currency.

To accomplish this we created a class called Fitcoin which defines our currency. This class handles all translations of the currency USD to Fitcoin and vice versa. As we expand our project we now have the opportunity to easily add other currencies to our currency object. Each user now has a wallet, which is created upon registration on the website. This wallet contains the amount of the currency the user has. The add balance method is triggered upon a balance purchase by a customer.